



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/682,012	10/09/2003	Ahhishek Kar	03-0984	7742
7590	11/13/2006		EXAMINER	
LSI Logic Corporation Legal Department - IP MS D-106 1621 Barber Lane Milpitas, CA 95035			PHAM, THAI V	
			ART UNIT	PAPER NUMBER
			2192	
			DATE MAILED: 11/13/2006	

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)
	10/682,012	KAR ET AL.
	Examiner Thai Van Pham	Art Unit 2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) Responsive to communication(s) filed on 10/09/2003.
 2a) This action is FINAL. 2b) This action is non-final.
 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4) Claim(s) 1-33 is/are pending in the application.
 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
 5) Claim(s) _____ is/are allowed.
 6) Claim(s) 1-33 is/are rejected.
 7) Claim(s) 5, 16 and 17 is/are objected to.
 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9) The specification is objected to by the Examiner.
 10) The drawing(s) filed on 09 October 2003 is/are: a) accepted or b) objected to by the Examiner.
 Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
 Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
 11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 a) All b) Some * c) None of:
 1. Certified copies of the priority documents have been received.
 2. Certified copies of the priority documents have been received in Application No. _____.
 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892)	4) <input type="checkbox"/> Interview Summary (PTO-413)
2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948)	Paper No(s)/Mail Date. _____
3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____	5) <input type="checkbox"/> Notice of Informal Patent Application
	6) <input type="checkbox"/> Other: _____

DETAILED ACTION

This is the initial office action based on the application filed on November 10, 2006.

Priority date that has been considered for this application is October 9, 2003. Claims 1

– 33 are currently pending and have been considered below.

Specification

1. The disclosure is objected to because of the following informalities: typographical errors.

-- On line 9 of page 10 of the Specification, “3” and “,” are missing in “...*including factory InvocationHandler instance 3 406, all inherit...*”.

-- On line 25 of page 10 of the Specification, “506” is mistyped as “505” in “...*the process passes to block **505 506**...*”.

-- On line 13 of page 12 of the Specification, “*thought*” is misspelled as “*though*” in “...*may be though thought of as including...*”.

Appropriate correction is required.

Claim Rejections - 35 USC § 101

35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

2. Claims 23 – 33 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

-- Claim 23: recites “*a computer program product*” that may include “transmission-type media”, i.e., signal – a form of energy (Specification, page 14: lines 13 – 18; “*...the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution...*”).

A product is a tangible physical article or object, some form of matter, which a signal is not. That the other product classes, machine and composition of matter, require physical matter is evidence that a manufacture was also intended to require physical matter. A signal, a form of energy, does not fall within either of the two definitions of manufacture. Thus, a signal does not fall within one of the four statutory classes of U.S.C. 101. (See Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility (See Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility (OG Cite: 1300 OG142), Annex IV(c)).

In the principle of compact prosecution, Examiner anticipates the claim will be amended to become statutory claim as such “*A computer program, recorded on a computer-readable medium, in a data processing system...*” or “*A computer program, embodied in a recordable-type media, in a data processing system...*”

Claims 24 – 33: are dependent claims of claim 23. These claims all fail to remedy the non-statutory issue, thus, also being rejected for the same reason above.

Claim Rejections - 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

3. Claims 5, 16, and 27 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

-- Claims 5, 16, and 27: recite the limitation "*said invocation instance*" in the last element of the claims. There is insufficient antecedent basis for this limitation in the claim(s). Examiner assumes the phrase is meant to be "*said invocation handler instance*" for further examination purpose.

Appropriate correction is required.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

4. Claims 1 – 33 are rejected under 35 U.S.C. 103(a) as being obvious over **Jones et al.** (US 6,877,163) in view of **McIntyre** (US 6,415,435).

-- Claim 1.

Jones discloses a *method in a data processing system that implements an object-oriented software environment* (Col. 1: lines 6 –11; "...object-oriented data processing system...") *for translating method calls to version-specific method calls, said method comprising the steps of:*

- *providing an interface to an underlying object, applications utilizing said interface to communicate with said underlying object, said interface being separate from said underlying object;*

(Col. 3: lines 59 – 66; "...the client, at runtime, specifies the interfaces in which it is interested..."

Col. 4: lines 21 – 35; "...dynamic proxy classes may be applied to remote method invocation ("RMI") ...the interfaces implemented by the dynamic proxy class might be a list of the methods available on the remote sever...")

- *generating a plurality of version-specific underlying objects, each one of said version-specific underlying objects being a different version of said underlying object;*

(Col. 3: lines 42 – 48; "...event generators, such as a windows manager and a device driver...." Version-specific objects are simply objects belonging to a specific type of object, such as a device driver, that are fundamentally the same but slightly different in some supported features. Thus, event generators encompass version-specific objects as well.)

- *generating a translation object for communicating between said interface and each one of said version-specific underlying objects;*

(Col. 3: lines 5 – 14; "...a proxy class, dynamically generated at runtime, that implements a list of interfaces specified at runtime. A method invocation through an interface on an instance of the class is encoded and dispatched uniformly regardless of the type of interface to an invocation handler object that performs the invocation of the requested method...". It's a matter of implementation choice to have a proxy class

object and its invocation handler object as separate object entities or linked into one. Thus, a translation object can be thought of as a combination of objects of a proxy class and its associated invocation handler.)

- *utilizing said translation object for translating an interface method call invoked on said interface to a version-specific method call for said underlying object for each version of said underlying object; and*

(Col. 3: lines 29 – 41; "...The proxy class instance automatically encodes and dispatches a method invocation of a method on an interface implemented by the proxy class instance to an invocation handler object that automatically handles the request and returns the result...")

Jones's disclosure solves the problem of handling a method invocation on an instance of a class through an interface, regardless of the type of that interface by means of implementation of a proxy class and invocation handler (Fig. 2, Col. 5: lines 6 – 21).

The proxy class disclosed by **Jones** can further support multiple interfaces for different types of objects as specified by the client.

Jones, however, does not explicitly disclose the method further comprising:

- *generating said translation object from a single proxy class and a single invocation handler class, wherein the same proxy class and invocation handler class are used to generate said translation object for each different version of said underlying object.*

McIntyre discloses a method for determining compatibility of parent classes in an object-oriented environment using versioning, comprising:

- *generating said translation object from a single proxy class and a single invocation handler class, wherein the same proxy class and invocation handler class are used to generate said translation object for each different version of said underlying object.*

(Fig. 3 – item 302: “Class Definition” and associate text, e.g., Col. 5: lines 12 – 40; “Versioning information” contained within a “class definition”. Compatible version-specific objects are created in form of a class hierarchy in which successor or child versions inherit properties of their predecessor or parent versions. Thus, as long as the versions of an object are compatible with respect to their fundamental properties, they are all generated from the same parent class. As a result, only a single interface with appropriate configuration parameters is necessary to handle method calls of the different versions of an object.)

In **McIntyre**’s disclosure, versions of a particular object are simply instances of classes in a class hierarchy originated from the same parent class. Thus, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of **McIntyre** that uses a single parent class for different object versions into **Jones**’s disclosure so that a specific set of proxy class and instance handler of **Jones** would generate different translation objects for the corresponding version-specific objects. The motivation for combining the two teachings above is to simplify the proxy implementation in such a way that it would utilize a single interface to handle all method invocations on different version-specific objects for the reason that these version-specific objects are fundamentally the same with small variations in

configurations, such as configuration parameters, which can be handled at the instant handler.

-- Claim 2.

Jones and **McIntyre** disclose *the method according to claim 1, further comprising the steps of.*

- *determining a current version of said underlying object;*

(**McIntyre**: Figs. 4 and 5 and associated text; “versioning information” located in parent and child “class definitions”.)

- *determining whether said translation object exists for said current version; and*

(**Jones**: Col. 6: line 63 – Col. 7: line 3; “...allow a proxy class to be generated at runtime that can implement interfaces specified at runtime thereby avoiding a need for pre-generation of the proxy class and predetermination of the interfaces before runtime.... ”

Jones: Col. 8: lines 1 – 4; “Proxy classes, as well as instances of them, may be created using the static methods of the class.” In static mode, proxy instance and invocation handler instance are created only when they have not existed.)

- *in response to determining that said translation object does not exist, generating said translation object utilizing a building object.*

(**Jones**: Col. 5: lines 5 – 14; “...a proxy class, dynamically generated at runtime, that implements a list of interfaces specified at runtime.... ” In dynamic mode, proxy class and invocation handler are always created at runtime.

Jones: Figs. 2 and 3, Col. 6: lines 7 – 21; "...The client then creates a proxy class instance 204 by calling the constructor of the proxy class 202 with the invocation handler 122 as an argument (step 314).")

-- Claim 3.

Jones and McIntyre disclose *the method according to claim 2, further comprising the steps of.*

- *creating said building object utilizing said proxy class and said invocation handler class.*

(**Jones**: Figs. 2 and 3, Col. 6: lines 7 – 21; "...The client then creates a proxy class instance 204 by calling the constructor of the proxy class 202 with the invocation handler 122 as an argument (step 314).")

-- Claim 4.

Jones and McIntyre disclose *the method according to claim 1, further comprising the steps of.*

- *generating a building instance of said proxy class and said invocation handler class;*

(**Jones**: Fig. 4, Col. 6: lines 23 – 42; "...the getProxyClass method creates the code for the constructor that will create an instance of the proxy class...")

- *utilizing said building instance to generate an invocation handler instance for said interface;*

(**Jones**: Fig. 4, Col. 6: lines 16 – 18: "...The client also creates the invocation handler 122 that will carry out the desired functionality of a call to an instance 204 of the proxy class...")

- *utilizing said building instance to generate said translation object; and*

(**Jones**: Figs. 2 and 3, Col. 6: lines 7 – 21; "...The client then creates a proxy class instance 204 by calling the constructor of the proxy class 202 with the invocation handler 122 as an argument (step 314).")

- *including said invocation handler instance in said translation object.*

(**Jones**: Figs. 2 and 3, Col. 6: lines 7 – 21; "...the proxy class 202 with the invocation handler 122 as an argument (step 314).")

-- Claim 5.

Jones and **McIntyre** disclose *the method according to claim 1, further comprising the steps of.*

- *generating a translation instance for a particular version of said underlying object;*

(**Jones**: Fig. 2 – items 204 and 122, Col. 5: lines 6 – 21; the invocation handler is implemented as a part of the proxy class.)

- *including an instance of said proxy class and an instance of said invocation handler class;*

(**Jones**: Figs. 2 and 3, Col. 6: lines 7 – 21; "...The client then creates a proxy class instance 204 by calling the constructor of the proxy class 202 with the invocation handler 122 as an argument (step 314).")

- *invoking said interface method call on said translation instance;*

(**Jones**: Fig. 5 – item 502, Col. 6: lines 43 – 62; “invoke method”.)

- *passing said interface method call from said proxy instance to said invocation handler instance; and*

(**Jones**: Fig. 5 – item 504, Col. 6: lines 43 – 62; “Proxy class instance encodes and dispatches method invocation to invocation handler”.)

- *invoking a version-specific method call on said particular version of said underlying object using said invocation handler instance, said version-specific method call being equivalent to said invoked interface method call.*

(**Jones**: Fig. 5 – item 506, Col. 6: lines 43 – 62; “invocation handler processes method invocation”.)

-- Claim 6.

Jones and **McIntyre** disclose *the method according to claim 5, further comprising the steps of*.

- *utilizing, by said invocation handler instance, said interface method call and original parameters included in said interface method call to locate said version-specific method call;*

(Fig. 3 – items 304 – 312, Col. 5: line 50 – Col. 6: line 22; invocation handler is created according to the specified interfaces as parameters at the client.)

- *passing said original parameters to said invoked version-specific method call;*

(**Jones**: Fig. 5 – item 506, Col. 6: lines 43 – 62; “invocation handler processes method invocation”.)

- *receiving, by said invocation handler instance, a return object from version-specific method call; and*

(Jones: Col. 6: lines 43 – 62; "...The invocation handler **122** process the method invocation using its predetermined functionality and obtains the result (step **506**)...".)

- *returning, by said invocation handler instance, said return object.*

(Jones: Fig. 5 – item 508, Col. 6: lines 43 – 62; "invocation handler returns result to the proxy class instance".)

-- Claim 7.

Jones and McIntyre disclose *the method according to claim 6, further comprising the steps of.*

- *invoking said interface method call on said translation instance by a client object; and*

(Jones: Fig. 5 – item 502, Col. 6: lines 43 – 62; "invoke method".)

- *returning, by said invocation handler instance, said return object to said client object.*

(Jones: Fig. 5 – item 510, Col. 6: lines 43 – 62; "proxy class instance returns result to client".)

-- Claim 8.

Jones and McIntyre disclose *the method according to claim 5, further comprising the steps of.*

- *utilizing, by said invocation handler instance, said interface method call and original parameters included in said interface method call to locate said version-specific method call;*

(**Jones**: Fig. 3 – items 304 – 312, Col. 5: line 50 – Col. 6: line 22; invocation handler is created according to the specified interfaces as parameters at the client.)

- determining whether any of said original parameters are proxy objects; and

(**Jones**: Fig. 4, Col. 6: lines 23 – 42; "...getProxyClass method invoked by the client ...verifies the arguments to make sure that they are valid (step 402)..."")

- in response to determining that at least one of said original parameters are proxy objects, replacing each said proxy object with an underlying object type for said particular version.

(**Jones**: Fig. 4, Col. 6: lines 23 – 42; "...getProxyClass method invoked by the client ...verifies the arguments to make sure that they are valid (step 402) ...getProxyClass method creates the code for the constructor that will create an instance 204 of the proxy class 202 (step 406).")

-- Claim 9.

Jones and McIntyre disclose *the method according to claim 7*,

- determining whether said return object is a second interface; and

(**Jones**: Fig. 4, Col. 6: lines 23 – 42; "...the client invokes this method with the Class Objects for the interface 206 and 208 to be implemented, the method verifies the arguments to make sure that they are valid (step 402)..."")

Jones and McIntyre, however, do not disclose the method *further comprising the steps of*.

- in response to a determination that said return object is said second interface, creating a second proxy instance that implements a return type for said second interface.

Jones's disclosure solves the problem of handling a method invocation on an instance of a class through an interface, regardless of the type of that interface by means of implementation of a proxy class and invocation handler (Fig. 2, Col. 5: lines 6 – 21). Furthermore, the proxy class disclosed by **Jones** can further support multiple interfaces for different types of objects as specified by the client. (Fig. 4, Col. 6: lines 23 – 42.) Thus, it would have been obvious to one of ordinary skill in the art at the time the invention was made to recognize that **Jones**'s invention could create a second proxy instance to handle a calls from a second interface so that each proxy class is specific to a particular interface for sake of simplicity in implementation of the proxy class.

-- Claim 10.

Jones and **McIntyre** disclose *the method according to claim 7, further comprising the steps of.*

- determining whether said return object is a second interface;

(**Jones**: Fig. 4, Col. 6: lines 23 – 42; "...the client invokes this method with the Class Objects for the interface **206** and **208** to be implemented, the method verifies the arguments to make sure that they are valid (step **402**)...")

Jones and **McIntyre**, however, do not disclose the method *further comprising the steps of.*

- in response to determining that said return object is said second interface, determining whether said return type is an array; and

- in response to determining that said return object is an array, creating a separate proxy instance that implements a return type for said second interface for each element of said array.

An array is simply a collection of data structures of the same type linked together by a memory address pointer. **Jones**'s disclosure solves the problem of handling a method invocation on an instance of a class through an interface, regardless of the type of that interface by means of implementation of a proxy class and invocation handler (Fig. 2, Col. 5: lines 6 – 21). Furthermore, the proxy class disclosed by **Jones** can further support multiple interfaces for different types of objects as specified by the client. (Fig. 4, Col. 6: lines 23 – 42.) The API method can store the interface references in an array of or separate memory location as a matter of implementation choices. Regardless of whether or not the return type of the object referring to the interfaces, **Jones**'s invention provides the proxy instance with the ability to handle method invocations at all of the interfaces.

Thus, it would have been obvious to one of ordinary skill in the art at the time the invention was made to recognize that **Jones**'s invention could create a second proxy instance for each of the interface object references returned by the API method to handle calls from individual interfaces so that each proxy class is specific to a particular interface for sake of simplicity in implementation of the proxy class.

-- Claim 11.

Jones and **McIntyre** disclose *the method according to claim 1, further comprising the steps of:*

- *said translation object being a single object for inserting translation code for translating said interface method call and for manipulating return parameters received after execution of said translated interface method call.*

(Fig. 5, Col. 6: lines 43 – 62; “...the proxy class instance 204 encodes and dispatches the method invocation to the invocation handler 122 (step 504). The invocation handler then process the method invocation using its predetermined functionality and obtains the result (step 506)...”. It’s a matter of implementation choice to have a proxy class object and its invocation handler object as separate object entities or linked into one. Thus, a translation object can be thought of as a combination of objects of a proxy class and its associated invocation handler.)

Claims 12 – 22: are system claims for performing a method corresponding to the method of claims 1 – 11, respectively; Therefore, claims 12 – 22 are rejected for the same reason set forth in connection to the rejection of claims 1 – 11 above, respectively.

Claims 23 – 33: are computer product claims for performing a method corresponding to the method of claims 1 – 11, respectively; Therefore, claims 23 – 33 are rejected for the same reason set forth in connection to the rejection of claims 1 – 11 above, respectively.

Conclusion

5. The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure. See the attached Notice of References Cited.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Thai Van Pham whose telephone number is (571) 270-1064. The examiner can normally be reached on Monday - Thursday, 8am - 3pm EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TVP
11/10/2006


TUAN DAM
SUPERVISORY PATENT EXAMINER